ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

Факультет безопасности информационных технологий
Кафедра проектирования и безопасности компьютерных систем

Дисциплина:
«Операционные системы»

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ №2

Выполнил:
Студент гр. N3247 Гаврилова В.В.

Проверил:
Ханов А. Р.

Санкт-Петербург
2021г.

Задание:

Membomb

1. Написать программу выделения памяти и заполнения ее нулями с шагом, равным размеру страницы памяти (mmap, VirtualAlloc)
2. Составить график свободной памяти
3. Ознакомиться с работой демона OOM Killer в Linux
4. Достичь сообщения о невозможности выделить память в Windows

Выполнение:
Для Windows

1) Программа выделения и заполнения памяти с помощью VirtualAlloc

```c
#include <windows.h>
#include <psapi.h>

typedef void (*ProcFunc)(DWORD pid);
#define ALLOC_SIZE 0x1000
LPVOID buf;

void ForEachProcess(ProcFunc f)
{
    DWORD aProcesses[1024], cbNeeded;
    if (!EnumProcesses( aProcesses, sizeof(aProcesses), &cbNeeded))
        return;
    for (unsigned int i = 0; i < cbNeeded / sizeof(DWORD); i++)
        if (aProcesses[i] != 0)
            f(aProcesses[i]);
}

void RemoteLeak(DWORD pid)
{
    HANDLE hProcess = OpenProcess( PROCESS_ALL_ACCESS, FALSE, pid );
    if (hProcess == NULL)
        return;
    for (;;)
    {
        LPVOID ptr = VirtualAllocEx(hProcess, NULL, ALLOC_SIZE, MEM_COMMIT, PAGE_READWRITE);
        if (ptr == NULL)
            return;
        WriteProcessMemory(hProcess, ptr, buf, ALLOC_SIZE, NULL);
    }
}

int main(void)
{
    buf = malloc(ALLOC_SIZE);
    if (buf == NULL)
        return 0;
    memset(buf, 0xFF, ALLOC_SIZE);
    ForEachProcess(RemoteLeak);
    return 0;
}
```

У нас есть программа с двумя функциями.
Сначала мы выделяем чанк памяти размером ALLOC_SIZE (4096). Этим чанком мы будем в дальнейшем заполнять память процессов. Malloc возвращает Null если объём памяти недоступен. С помощью memset заполняем буфер символами 0xFF.
В первой функции (forEachProcess) мы получаем идентификаторы всех процессов: с помощью функции над процессами из psapi.h помещаем иды процессов в массив. Пробегаемся по массиву и подаем ненулевые пиды в качестве аргумента второй функции.
Во второй функции (RemoteLeak) мы получаем дескриптор процесса по пиду в аргументе, а затем с помощью VirtualAllocEx и бесконечного цикла выделяем память в нём, после чего заполняем её буфером с помощью WriteProcessMemory.

2) Графики:

График использования памяти в момент выполнения файла .exe

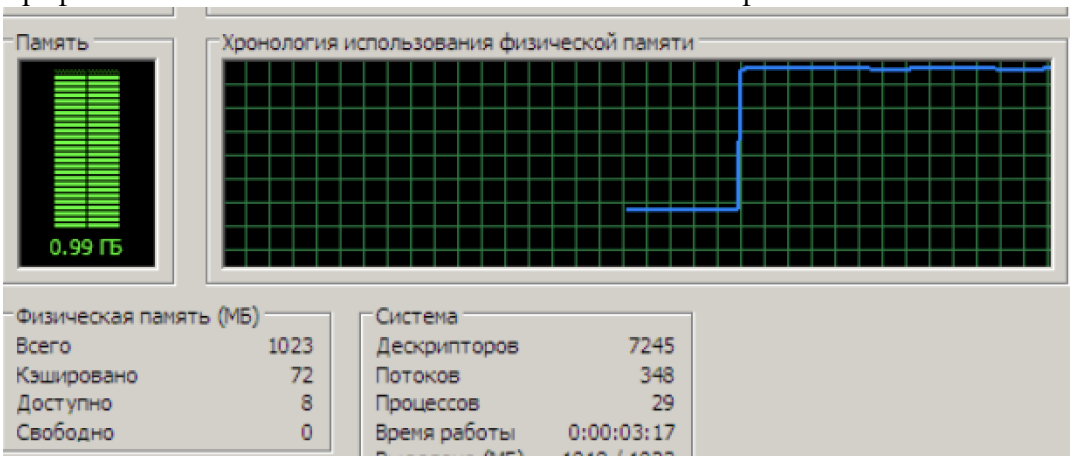
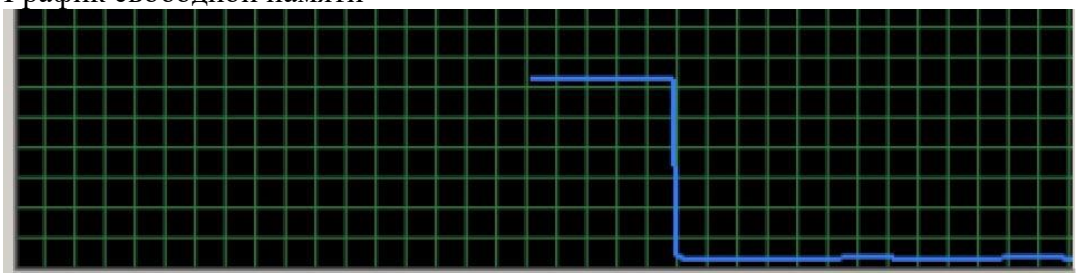
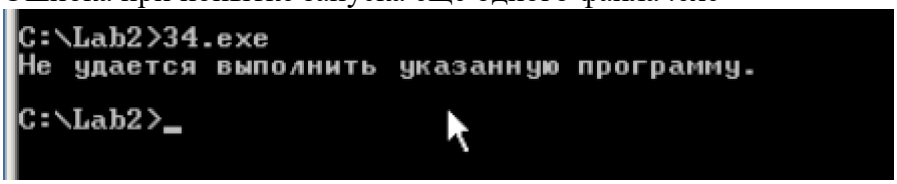


График свободной памяти

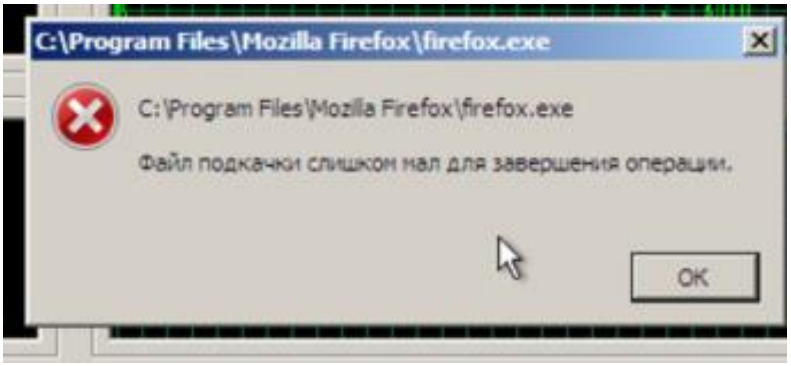


3) Достичь сообщения о невозможности выделить память в Windows оказалось невозможным:

Ошибка при попытке запуска еще одного файла .exe



Ошибка при попытке запуска браузера:



Для Linux:

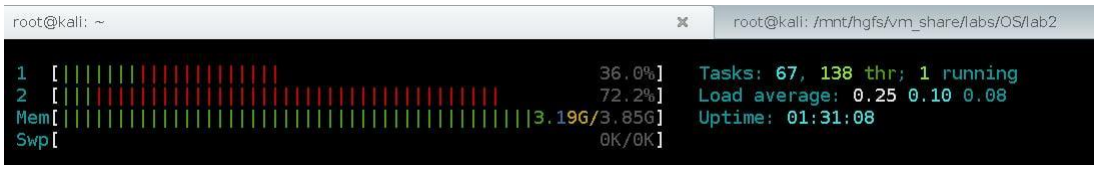
Программа выделения и заполнения памяти с помощью mmap

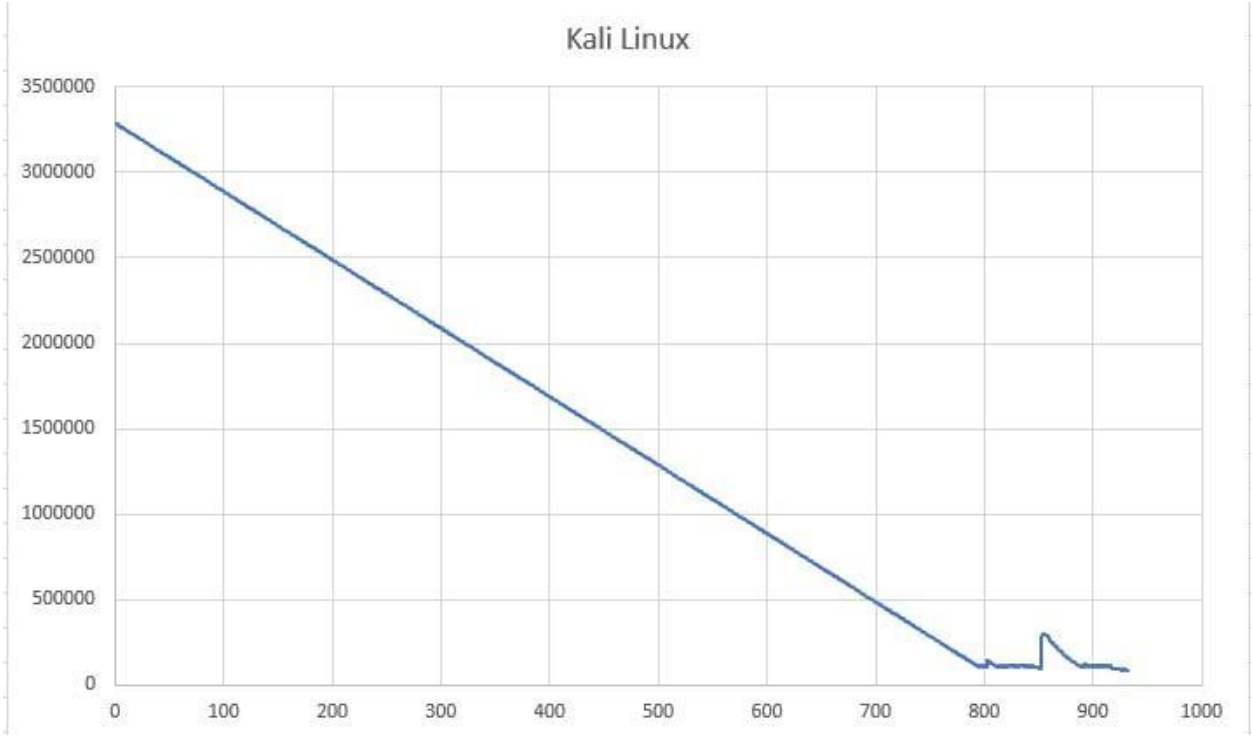
```c
#define _GNU_SOURCE

#include <unistd.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <stdio.h>
#include <sys/resource.h>
#include <signal.h>

int main(){

        if (setresuid(0, 0, 0) == -1) {           //Выставить реальный, эффективный юзер айди на рута
                printf("Are you root?!\n");
                return 1;
        }

        sigset_t mask;
        sigfillset(&mask);
        sigprocmask(SIG_SETMASK, &mask, NULL);   // Блокировать все сигналы кроме SIGKILL и SIGSTOP

        struct rlimit memory = { RLIM_INFINITY, RLIM_INFINITY }, // Позволим процессу вирт. аллоцировать бесконечный объем памяти
                      signal = { RLIM_INFINITY, RLIM_INFINITY};  // Бесконечное кол-во сигналов к процессу ставить в очередь
        setrlimit(RLIMIT_AS, &memory);
        setrlimit(RLIMIT_SIGPENDING, &signal);


        char proc_name[20];
        sprintf(proc_name, "/proc/%u/oom_adj", getpid());
        FILE* oom_killer_file = fopen(proc_name, "w");
        if (oom_killer_file){
                fprintf(oom_killer_file, "-17\n");  // Говорим OOM киллеру игнорировать наш процесс
                fclose(oom_killer_file);
        }



        int cnt=0;
        while (1){       // Сама бомба

                if (cnt % 1000 == 0) system("cat /proc/meminfo | grep MemFree | awk '{print $2}' >> logsmem.txt");
                char* ad = mmap(0, 0x1000, 6, 0x22, -1 ,0);
                if (ad) ad[0]=0;
                else printf("ERR");
                cnt++;
        }
        return 0;

}
```

Если написать только сам цикл, без предварительных строк кода, то membomb займет всю память и процесс будет убит киллером.
Ведь когда процесс хочет аллоцировать очередной объем памяти, но она кончилась - просыпается OOM киллер и убивает процесс, который набрал больше всего очков виновности.
Вот примерно так идёт подсчёт очков:

1. Считаем RSS процесса
2. Добавляем RSS всех дочерних процессов
3. Если процесс долго живёт, то значение уменьшается
4. Если у процесса niceness больше 0, то значение увеличивается.
5. Если есть флаги CAP_SYS_ADMIN или CAP_SYS_RAWIO, результат уменьшается
6. Смотрится значение /proc/<pid>/oom_adj, которое может задавать пользователь, чтобы повышать сопротивляемость OOM Killer'y.

График


root@kali: ~
root@kali: /mnt/hgfs/vm_share/labs/OS/lab2
1 [36.0%]
2 [72.2%]
Mem[3.19G/3.85G]
Swp[0K/0K]
Tasks: 67, 138 thr; 1 running
Load average: 0.25 0.10 0.08
Uptime: 01:31:08


Kali Linux
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
0
100
200
300
400
500
600
700
800
900
1000